УО «Белорусский государственный университет информатики и радиоэлектроники»

Кафедра ПОИТ

Отчет по лабораторной работе №3.2

по предмету «Основы алгоритмизации и программирования»

Вариант 20

Выполнил:

Захвей И.В.

Гр. 351005

Проверил:

Данилова Г. В.

Минск 2023

**Задание:**

Имеется множество, содержащее натуральные числа из некоторого

диапазона. Сформировать два множества, первое из которых содержит

все простые числа из данного множества, а второе все составные.

**Код программы Delphi:**

Program Project4;

Uses

System.SysUtils;

Type

TSet = Set of Byte;

TBorderArr = Array [0 .. 1] of Byte;

ERRORS\_CODE = (SUCCESS, INCORRECT\_DATA, EMPTY\_LINE, NOT\_TXT, FILE\_NOT\_EXIST,

INCORRECT\_DATA\_FILE, A\_LOT\_OF\_DATA\_FILE, FILE\_NOT\_AVAILABLE,

OUT\_OF\_BORDER, INCORRECT\_BORDERS);

Const

MAX\_NUMB = 255;

MIN\_NUMB = 0;

ERRORS: Array [ERRORS\_CODE] Of String = ('Successfull',

'Data is not correct',

'Line is empty, please be careful',

'This is not a .txt file',

'This file is not exist',

'Data in file is not correct',

'There are two numbers in file should be',

'File is can not be opened',

'Out of border [0, 255]',

'Incorrect borders');

Procedure PrintInf();

Begin

Writeln('Program forms two sets, the first of which contains all simple ',

#13#10,'numbers from this set, and the second contains others');

Writeln('Borders and numbers should be in the interval [0, 255]');

End;

Function IsNumbSimple(Numb: Integer): Boolean;

Var

IsSimple: Boolean;

RightBord, I: Integer;

Begin

RightBord := Trunc(Sqrt(Numb));

IsSimple := True;

if Numb > 3 then

For I := 2 To RightBord Do

If Numb Mod I = 0 Then

IsSimple := False;

IsNumbSimple := IsSimple;

End;

Function GetSetOfSimple(DefaultSet: TSet): TSet;

Var

SimpleSet: TSet;

Numb: Byte;

Begin

SimpleSet := [];

for Numb in DefaultSet do

If IsNumbSimple(Numb) then

Include(SimpleSet, Numb);

GetSetOfSimple := SimpleSet;

End;

Function GetSetOfComposit(DefaultSet, SimpleSet: TSet): TSet;

Var

CompositSet: TSet;

Begin

CompositSet := [];

CompositSet := DefaultSet - SimpleSet;

GetSetOfComposit := CompositSet;

End;

Function CreateSetWhithBorders(Var Borders: TBorderArr): TSet;

Var

NumSet: TSet;

I: Byte;

Begin

NumSet := [];

for I := Borders[0] to Borders[1] do

Include(NumSet, I);

CreateSetWhithBorders := NumSet;

End;

Function InpChoice(Var Choice: Integer): ERRORS\_CODE;

Var

Err: ERRORS\_CODE;

ChoiceStr: String;

Begin

Err := SUCCESS;

Readln(ChoiceStr);

If (ChoiceStr = '1') Or (ChoiceStr = '2') Then

Choice := StrToInt(ChoiceStr)

Else If (Length(ChoiceStr) > 0) Then

Err := INCORRECT\_DATA

Else

Err := EMPTY\_LINE;

InpChoice := Err;

End;

Function InpValidBorder(Var Numb: Byte): ERRORS\_CODE;

Var

Err: ERRORS\_CODE;

Line: String;

IsCorrect: Boolean;

NumbInt: Integer;

Begin

Err := SUCCESS;

IsCorrect := True;

Readln(Line);

Try

NumbInt := StrToInt(Line);

Except

Err := INCORRECT\_DATA;

IsCorrect := False;

End;

if IsCorrect then

If (NumbInt > MAX\_NUMB) Or (NumbInt < MIN\_NUMB) Then

Err := OUT\_OF\_BORDER

Else

Numb := NumbInt;

InpValidBorder := Err;

End;

Function InpValidBorders(Var Borders: TBorderArr): ERRORS\_CODE;

Var

Err: ERRORS\_CODE;

Line: String;

IsCorrect: Boolean;

Begin

Err := InpValidBorder(Borders[0]);

if Err = SUCCESS then

Begin

Err := InpValidBorder(Borders[1]);

if (Err = SUCCESS) And (Borders[0] > Borders[1]) then

Err := INCORRECT\_BORDERS;

End;

InpValidBorders := Err;

End;

Function InputFromConsole(): TBorderArr;

Var

Err: ERRORS\_CODE;

Borders: TBorderArr;

Begin

Writeln('Enter the borders through the Enter');

Repeat

Err := InpValidBorders(Borders);

If (Err <> SUCCESS) then

Writeln(ERRORS[Err], #10#13, 'Please, enter again');

Until (Err = SUCCESS);

InputFromConsole := Borders;

End;

Function UserChoice(): Integer;

Var

Choice: Integer;

Err: ERRORS\_CODE;

Begin

Writeln('Choose a way of input/output of data', #13#10, '1 -- Console',

#13#10, '2 -- File');

Repeat

Err := InpChoice(Choice);

If (Err <> SUCCESS) then

Writeln(ERRORS[Err], #13#10, 'Please, enter again');

Until (Err = SUCCESS);

UserChoice := Choice;

End;

Function FileAvailable(Name: String; ForReset: Boolean): ERRORS\_CODE;

Var

Err: ERRORS\_CODE;

MyFile: TextFile;

Begin

Err := SUCCESS;

AssignFile(MyFile, Name);

If ForReset Then

Try

Try

Reset(MyFile);

Finally

CloseFile(MyFile);

End;

Except

Err := FILE\_NOT\_AVAILABLE;

End

Else

Try

Try

Rewrite(MyFile);

Finally

CloseFile(MyFile);

End;

Except

Err := FILE\_NOT\_AVAILABLE;

End;

FileAvailable := Err;

End;

Function GetLastFourChar(Line: String): String;

Var

Start, I, Size: Integer;

LastFourChar: String;

Begin

Size := Length(Line);

Start := Size - 3;

For I := Start To Size Do

LastFourChar := LastFourChar + Line[I];

GetLastFourChar := LastFourChar;

End;

Function FileTxt(Name: String): ERRORS\_CODE;

Var

Err: ERRORS\_CODE;

LastFourChar: String;

Begin

Err := SUCCESS;

If Length(Name) > 4 Then

Begin

LastFourChar := GetLastFourChar(Name);

If LastFourChar <> '.txt' Then

Err := NOT\_TXT;

End

Else

Err := NOT\_TXT;

FileTxt := Err;

End;

Function FileExist(Name: String): ERRORS\_CODE;

Var

Err: ERRORS\_CODE;

Begin

Err := SUCCESS;

If Not FileExists(Name) Then

Err := FILE\_NOT\_EXIST;

FileExist := Err;

End;

Function GetFileName(ForReset: Boolean): String;

Var

IsCorrect: Boolean;

ErrExist, ErrTxt, ErrAvailable: ERRORS\_CODE;

FileName: String;

Begin

Repeat

IsCorrect := True;

Readln(FileName);

ErrExist := FileExist(FileName);

ErrTxt := FileTxt(FileName);

If (ErrExist <> SUCCESS) Then

Begin

Writeln(ERRORS[ErrExist]);

IsCorrect := False;

Writeln('Please, enter full path again');

End

Else If (ErrTxt <> SUCCESS) Then

Begin

Writeln(ERRORS[ErrTxt]);

IsCorrect := False;

Writeln('Please, enter full path again');

End

Else

Begin

ErrAvailable := FileAvailable(FileName, ForReset);

If (ErrAvailable <> SUCCESS) Then

Begin

Writeln(ERRORS[ErrAvailable]);

IsCorrect := False;

Writeln('Please, enter full path again');

End;

End;

Until IsCorrect;

GetFileName := FileName;

End;

Function ReadOneFromFile(Var Numb: Byte; Var MyFile: TextFile): ERRORS\_CODE;

Var

Line: String;

Err: ERRORS\_CODE;

IsCorrect: Boolean;

NumbInt: Integer;

Begin

Err := SUCCESS;

IsCorrect := True;

Try

Read(MyFile, NumbInt);

Except

Err := INCORRECT\_DATA\_FILE;

IsCorrect := False;

End;

if IsCorrect then

If (NumbInt > MAX\_NUMB) Or (NumbInt < MIN\_NUMB) Then

Err := OUT\_OF\_BORDER

Else

Numb := NumbInt;

ReadOneFromFile := Err;

End;

Function ReadFile(Var Borders: TBorderArr; Name: String): ERRORS\_CODE;

Var

Err: ERRORS\_CODE;

InfFile: TextFile;

Begin

AssignFile(InfFile, Name);

Reset(InfFile);

Err := ReadOneFromFile(Borders[0], InfFile);

if Err = SUCCESS then

Begin

Err := ReadOneFromFile(Borders[1], InfFile);

if Err = SUCCESS then

if (Borders[0] > Borders[1]) then

Err := INCORRECT\_BORDERS;

If Not EoF(InfFile) Then

Err := A\_LOT\_OF\_DATA\_FILE;

End;

CloseFile(InfFile);

ReadFile := Err;

End;

Function InputFromFile(): TBorderArr;

Var

Err: ERRORS\_CODE;

FileName: String;

Borders: TBorderArr;

Begin

Writeln('Enter full path to file');

Repeat

FileName := GetFileName(True);

Err := ReadFile(Borders, FileName);

If (Err <> SUCCESS) then

Writeln(ERRORS[Err], #13#10, 'Enter full path to file');

Until (Err = SUCCESS);

Writeln('Reading is successfull');

InputFromFile := Borders;

End;

Function InputInf(): TBorderArr;

Var

Borders: TBorderArr;

Choice: Integer;

Begin

Choice := UserChoice();

If (Choice = 1) Then

Borders := InputFromConsole()

Else

Borders := InputFromFile();

InputInf := Borders;

End;

Procedure OutputInConsole(Default, SimpleSet, CompositSet: TSet);

Var

Numb: Byte;

Begin

Write('Default set', #13#10, '{ ');

for Numb in Default do

Write(Numb, ' ');

Write('}', #13#10, 'Set with simple numbers', #13#10, '{ ');

for Numb in SimpleSet do

Write(Numb, ' ');

Write('}', #13#10, 'Set with composit numbers', #13#10, '{ ');

for Numb in CompositSet do

Write(Numb, ' ');

Writeln('}');

End;

Procedure OutputInFile(Default, SimpleSet, CompositSet: TSet);

Var

FileName: String;

Numb: Byte;

MyFile: TextFile;

Begin

Writeln('Enter full path to file');

FileName := GetFileName(False);

AssignFile(MyFile, FileName);

Rewrite(MyFile);

Write(MyFile, 'Default set', #13#10, '{ ');

for Numb in Default do

Write(MyFile, Numb, ' ');

Write(MyFile, '}', #13#10, 'Set with simple numbers', #13#10, '{ ');

for Numb in SimpleSet do

Write(MyFile, Numb, ' ');

Write(MyFile, '}', #13#10, 'Set with composit numbers', #13#10, '{ ');

for Numb in CompositSet do

Write(MyFile, Numb, ' ');

Writeln(MyFile, '}');

CloseFile(MyFile);

Writeln('Writing is successfull');

End;

Procedure OutputInf(Default, SimpleSet, CompositSet: TSet);

Var

Choice: Integer;

Begin

Choice := UserChoice();

If (Choice = 1) Then

OutputInConsole(Default, SimpleSet, CompositSet)

Else

OutputInFile(Default, SimpleSet, CompositSet);

End;

Var

Borders: TBorderArr;

Default, SetWithSimple, SetWithComposit: TSet;

Begin

PrintInf();

Borders := InputInf();

Default := CreateSetWhithBorders(Borders);

SetWithSimple := GetSetOfSimple(Default);

SetWithComposit := GetSetOfComposit(Default, SetWithSimple);

OutputInf(Default, SetWithSimple, SetWithComposit);

Readln;

End.

**Код программы С++:**

#include <iostream>

#include <string>

#include <fstream>

#include <set>

using namespace std;

enum ErrorsCode

{

SUCCESS,

INCORRECT\_DATA,

EMPTY\_LINE,

NOT\_TXT,

FILE\_NOT\_EXIST,

A\_LOT\_OF\_DATA\_FILE,

OUT\_OF\_BORDER,

INCORRECT\_BORDERS

};

const int MIN\_NUMB = 0,

MAX\_NUMB = 255;

const string ERRORS[] = { "Successfull",

"Data is not correct, or number is too large\n",

"Line is empty, please be careful\n",

"This is not a .txt file\n",

"This file is not exist\n",

"There is only one line in file should be\n",

"Out of border [0, 255]\n",

"Incorrect borders\n"};

void printInf()

{

cout << "Program forms two sets, the first of which contains all simple"

<< "\nnumbers from this set, and the second contains others"

<< "\nBorders and numbers should be in the interval [0, 255]\n";

}

bool isNumbSimple(int numb)

{

bool isSimple;

isSimple = true;

int rightBord;

rightBord = trunc(sqrt(numb)) + 1;

if (numb > 3)

for (int i = 2; i < rightBord && isSimple; i++)

if (numb % i == 0)

isSimple = false;

return isSimple;

}

static set<int> getSetOfSimple(set<int> defaultSet)

{

set<int> simpleSet;

for (int numb : defaultSet)

if (isNumbSimple(numb))

simpleSet.insert(numb);

return simpleSet;

}

static set<int> getSetOfComposit(set<int> defaultSet, set<int> simpleSet)

{

set<int> compositSet;

for (int numb : defaultSet)

if (!simpleSet.count(numb))

compositSet.insert(numb);

return compositSet;

}

void freeArr(int\* arr)

{

delete[] arr;

}

static set<int> createSetWithBorders(int\* borders)

{

set<int> numbSet;

borders[1]++;

int i;

for (i = borders[0]; i < borders[1]; i++)

numbSet.insert(i);

freeArr(borders);

return numbSet;

}

ErrorsCode inpChoice(int& choice)

{

ErrorsCode err;

string choiceStr;

err = SUCCESS;

getline(cin, choiceStr);

if (choiceStr == "1" || choiceStr == "2")

choice = stoi(choiceStr);

else

err = choiceStr.length() > 0 ? INCORRECT\_DATA : EMPTY\_LINE;

return err;

}

int userChoice()

{

int choice;

cout << "Choose a way of input/output of data\n"

<< "1 -- Console\n"

<< "2 -- File\n";

ErrorsCode err;

do

{

err = inpChoice(choice);

if (err != SUCCESS)

cout << ERRORS[err] << "Please, enter again\n";

} while (err != SUCCESS);

return choice;

}

ErrorsCode inpValidBorder(int& numb)

{

ErrorsCode err;

bool isCorrect;

err = SUCCESS;

isCorrect = true;

cin >> numb;

if (cin.fail())

{

cin.clear();

while(cin.get() != '\n');

err = INCORRECT\_DATA;

isCorrect = false;

}

if (isCorrect && cin.get() != '\n')

{

while(cin.get() != '\n');

err = INCORRECT\_DATA;

isCorrect = false;

}

if (isCorrect && (numb > MAX\_NUMB) || (numb < MIN\_NUMB))

err = OUT\_OF\_BORDER;

return err;

}

ErrorsCode inpValidBorders(int\* borders)

{

ErrorsCode err;

err = inpValidBorder(borders[0]);

if (err == SUCCESS)

{

err = inpValidBorder(borders[1]);

if (err == SUCCESS && borders[0] > borders[1])

err = INCORRECT\_BORDERS;

}

return err;

}

int\* inputFromConsole()

{

cout << "Enter the borders through the Enter\n";

int err;

int\* borders = new int[2];

do

{

err = inpValidBorders(borders);

if (err != SUCCESS)

cout << ERRORS[err] << "Please, enter again\n";

} while (err != SUCCESS);

return borders;

}

ErrorsCode readOneFromFile(int& numb, ifstream& file)

{

ErrorsCode err;

bool isCorrect;

isCorrect = true;

err = SUCCESS;

file >> numb;

if (file.fail())

{

file.clear();

err = INCORRECT\_DATA;

isCorrect = false;

}

if (isCorrect && (numb > MAX\_NUMB) || (numb < MIN\_NUMB))

err = OUT\_OF\_BORDER;

return err;

}

ErrorsCode readFile(int\* borders, string fileName)

{

ErrorsCode err = SUCCESS;

bool isCorrect = true;

ifstream file(fileName);

err = readOneFromFile(borders[0], file);

if (err == SUCCESS)

{

err = readOneFromFile(borders[1], file);

if (err == SUCCESS)

{

if (borders[0] > borders[1])

err = INCORRECT\_BORDERS;

if (!file.eof())

err = A\_LOT\_OF\_DATA\_FILE;

}

}

file.close();

return err;

}

ErrorsCode isFileExist(string nameOfFile)

{

ErrorsCode err;

ifstream file(nameOfFile);

err = file.is\_open() ? SUCCESS : FILE\_NOT\_EXIST;

file.close();

return err;

}

string getLastFourChar(string line)

{

string lastFourChar;

int start, i, size;

size = line.length();

start = size - 4;

for (i = start; i < size; i++)

lastFourChar += line[i];

return lastFourChar;

}

ErrorsCode thisIsTxtFile(string& fileName)

{

ErrorsCode err = SUCCESS;

string lastFourChar;

if (fileName.length() > 4)

{

lastFourChar = getLastFourChar(fileName);

if (lastFourChar != ".txt")

err = NOT\_TXT;

}

else

err = NOT\_TXT;

return err;

}

string getFileName()

{

bool isIncorrect;

string name;

int errExist, errTxt;

cout << "Enter full path to file\n";

do

{

getline(cin, name);

errExist = isFileExist(name);

errTxt = thisIsTxtFile(name);

isIncorrect = false;

if (errTxt > 0)

{

cout << ERRORS[errTxt];

isIncorrect = true;

}

else if (errExist > 0)

{

cout << ERRORS[errExist];

isIncorrect = true;

}

} while (isIncorrect);

return name;

}

int\* inputFromFile()

{

string fileName;

int err;

int\* borders = new int[2];

do

{

fileName = getFileName();

err = readFile(borders, fileName);

if (err != SUCCESS)

cout << ERRORS[err] << "Please, enter full path again\n";

} while (err != SUCCESS);

cout << "Reading is successfull\n";

return borders;

}

int\* inputInf()

{

int\* borders;

int choice = userChoice();

if (choice == 1)

borders = inputFromConsole();

else

borders = inputFromFile();

return borders;

}

void outputInConsole(set<int> defaultSet, set<int> setSimple, set<int> setComposit)

{

cout << "Default set\n{ ";

for (int i : defaultSet)

cout << i << " ";

cout << "}\nSet with simple numbers\n{ ";

for (int i : setSimple)

cout << i << " ";

cout << "}\nSet with composite numbers\n{ ";

for (int i : setComposit)

cout << i << " ";

cout << "}";

}

void outputInFile(set<int> defaultSet, set<int> setSimple, set<int> setComposit)

{

string fileName = getFileName();

ofstream file(fileName);

file << "Default set" << endl << "{ ";

for (int i : defaultSet)

file << i << " ";

file << "}" << endl << "Set with simple numbers" << endl << "{ ";

for (int i : setSimple)

file << i << " ";

file << "}" << endl << "Set with composite numbers" << endl << "{ ";

for (int i : setComposit)

file << i << " ";

file << "}";

cout << "Writing is successfull\n";

file.close();

}

void outputInf(set<int> defaultSet, set<int> setSimple, set<int> setComposit)

{

int choice = userChoice();

if (choice == 1)

outputInConsole(defaultSet, setSimple, setComposit);

else

outputInFile(defaultSet, setSimple, setComposit);

}

int main()

{

int\* borders;

set<int> defaultSet, setWithSimple, setWithComposit;

printInf();

borders = inputInf();

defaultSet = createSetWithBorders(borders);

setWithSimple = getSetOfSimple(defaultSet);

setWithComposit = getSetOfComposit(defaultSet, setWithSimple);

outputInf(defaultSet, setWithSimple, setWithComposit);

}

**Код программы Java:**

import java.io.File;

import java.io.IOException;

import java.io.PrintWriter;

import java.nio.file.Path;

import java.nio.file.Paths;

import java.util.Scanner;

import java.util.TreeSet;

public class Main {

static enum Codes {

SUCCESS,

INCORRECT\_DATA,

EMPTY\_LINE,

NOT\_TXT,

FILE\_NOT\_EXIST,

INCORRECT\_DATA\_FILE,

A\_LOT\_OF\_DATA\_FILE,

IN\_OUT\_FILE\_EXCEPTION,

OUT\_OF\_BORDER,

INCORRECT\_BORDERS;

}

static final int MAX\_NUMB = 255,

MIN\_NUMB = 0;

static final String[] ERRORS ={"Successfull",

"Data is not correct, or number is too large",

"Line is empty, please be careful",

"This is not a .txt file",

"This file is not exist",

"Data in file is not correct",

"There are two numbers in file should be",

"Exception with output/input from the file",

"Out of border [0, 255]",

"Incorrect borders"};

static void printInf() {

System.out.println("Program forms two sets, the first of which contains all

simple " + "\nnumbers from this set, and the second contains others");

System.out.println("Borders and numbers should be in the interval [0, 255]");

}

static boolean isNumbSimple(int numb) {

boolean isSimple;

isSimple = true;

int rightBord;

rightBord = (int) Math.sqrt(numb)+1;

if (numb > 3)

for (int i = 2; i < rightBord && isSimple; i++)

if (numb % i == 0)

isSimple = false;

return isSimple;

}

static TreeSet<Integer> getSetOfSimple(TreeSet<Integer> defaultSet) {

TreeSet<Integer> simpleSet = new TreeSet<>();

for (int numb : defaultSet)

if (isNumbSimple(numb))

simpleSet.add(numb);

return simpleSet;

}

static TreeSet<Integer> getSetOfComposit(TreeSet<Integer> defaultSet,

TreeSet<Integer> simpleSet) {

TreeSet<Integer> compositSet = new TreeSet<>();

for(int numb : defaultSet)

if (!simpleSet.contains(numb))

compositSet.add(numb);

return compositSet;

}

static TreeSet<Integer> createSetWithBorders(int[] borders) {

TreeSet<Integer> numbSet = new TreeSet<>();

borders[1]++;

int i;

for (i = borders[0]; i < borders[1]; i++)

numbSet.add(i);

return numbSet;

}

static Codes inputChoice(Scanner input, int[] choice){

Codes err;

String choiceStr;

err = Codes.SUCCESS;

choiceStr = input.nextLine();

if (choiceStr.equals("1") || choiceStr.equals("2")) {

choice[0] = Integer.parseInt(choiceStr);

} else {

err = choiceStr.isEmpty() ? Codes.EMPTY\_LINE : Codes.INCORRECT\_DATA;

}

return err;

}

static int userChoice(Scanner input) {

int[] choice = {0};

System.out.println("Choose a way of input/output of data\n"

+ "1 -- Console\n"

+ "2 -- File");

Codes err;

do {

err = inputChoice(input, choice);

if (err != Codes.SUCCESS) {

System.err.println(ERRORS[err.ordinal()]);

System.out.println("Please, enter again");

}

} while (err != Codes.SUCCESS);

return choice[0];

}

static Codes inpValidBorder(int[] numb, Scanner input) {

int numbInt;

Codes err;

boolean isCorrect;

err = Codes.SUCCESS;

isCorrect = true;

numbInt = -1;

try {

numbInt = Integer.parseInt(input.nextLine());

} catch (NumberFormatException e) {

err = Codes.INCORRECT\_DATA;

isCorrect = false;

}

if (isCorrect)

if (numbInt > MAX\_NUMB || numbInt < MIN\_NUMB)

err = Codes.OUT\_OF\_BORDER;

else

numb[0] = numbInt;

return err;

}

static Codes inpValidBorders(int[] borders, Scanner input) {

Codes err;

int[] border = {0};

border[0] = borders[0];

err = inpValidBorder(border, input);

if (err == Codes.SUCCESS) {

borders[0] = border[0];

border[0] = borders[1];

err = inpValidBorder(border, input);

if (err == Codes.SUCCESS) {

borders[1] = border[0];

if (borders[0] > borders[1])

err = Codes.INCORRECT\_BORDERS;

}

}

return err;

}

static int[] inputFromConsole(Scanner input) {

int[] borders = {0,0};

System.out.println("Enter the borders through the Enter");

Codes err;

do {

err = inpValidBorders(borders, input);

if (err != Codes.SUCCESS) {

System.err.println(ERRORS[err.ordinal()]);

System.out.println("Please, enter again");

}

} while (err != Codes.SUCCESS);

return borders;

}

static Codes readOneFromFile(int[] numb, Scanner file) {

Codes err;

int numbInt;

boolean isCorrect;

err = Codes.SUCCESS;

numbInt = 0;

isCorrect = true;

try {

numbInt = Integer.parseInt(file.next());

} catch (NumberFormatException e) {

err = Codes.INCORRECT\_DATA\_FILE;

isCorrect = false;

}

if (isCorrect)

if ((numbInt > MAX\_NUMB) || (numbInt < MIN\_NUMB))

err = Codes.OUT\_OF\_BORDER;

else

numb[0] = numbInt;

return err;

}

static Codes readFile(int[] borders, String fileName) throws IOException{

Codes err;

Path path = Paths.get(fileName);

Scanner file = new Scanner(path);

int[] border = {0};

border[0] = borders[0];

err = readOneFromFile(border, file);

if (err == Codes.SUCCESS) {

borders[0] = border[0];

border[0] = borders[1];

err = readOneFromFile(border, file);

if (err == Codes.SUCCESS) {

borders[1] = border[0];

if (borders[0] > borders[1])

err = Codes.INCORRECT\_BORDERS;

if (file.hasNextLine())

err = Codes.A\_LOT\_OF\_DATA\_FILE;

}

}

return err;

}

static Codes isFileExist(String fileName) {

File file = new File(fileName);

Codes err;

err = file.exists() ? Codes.SUCCESS : Codes.FILE\_NOT\_EXIST;

return err;

}

static Codes thisIsTxtFile(String fileName) {

Codes err;

err = fileName.endsWith(".txt") ? Codes.SUCCESS : Codes.NOT\_TXT;

return err;

}

static String getFileName(Scanner input) {

boolean isIncorrect;

String fileName;

Codes errTxt, errExist;

System.out.println("Enter full path to file");

do {

isIncorrect = false;

fileName = input.nextLine();

errTxt = thisIsTxtFile(fileName);

errExist = isFileExist(fileName);

if (errTxt != Codes.SUCCESS) {

isIncorrect = true;

System.err.println(ERRORS[errTxt.ordinal()]);

}

else if (errExist != Codes.SUCCESS) {

isIncorrect = true;

System.err.println(ERRORS[errExist.ordinal()]);

}

} while (isIncorrect);

return fileName;

}

static int[] inputFromFile(Scanner input){

Codes err;

int[] borders = {0, 0};

do {

String fileName = getFileName(input);

try {

err = readFile(borders, fileName);

} catch (IOException e) {

err = Codes.IN\_OUT\_FILE\_EXCEPTION;

}

if (err != Codes.SUCCESS) {

System.err.println(ERRORS[err.ordinal()]);

System.out.println("Please, enter full path again");

}

} while (err != Codes.SUCCESS);

System.out.println("Reading is successfull");

return borders;

}

static int[] inputInf(Scanner input){

int choice = userChoice(input);

int[] borders = {0, 0};

if (choice == 1) {

borders = inputFromConsole(input);

} else {

borders = inputFromFile(input);

}

return borders;

}

static void writeInConsole(TreeSet<Integer> defaultSet, TreeSet<Integer> setSimple,

TreeSet<Integer> setComposit) {

System.out.print("Default set\n{ ");

for (int i : defaultSet)

System.out.print(i + " ");

System.out.print("}\nSet with simple numbers\n{ ");

for (int i : setSimple)

System.out.print(i + " ");

System.out.print("}\nSet with composite numbers\n{ ");

for (int i : setComposit)

System.out.print(i + " ");

System.out.println("}");

}

static void writeInFile(TreeSet<Integer> defaultSet, TreeSet<Integer> setSimple,

TreeSet<Integer> setComposit, Scanner input) {

boolean isIncorrect;

do {

String fileName = getFileName(input);

isIncorrect = false;

try (PrintWriter file = new PrintWriter(fileName)) {

file.print("Default set\n{ ");

for (int i : defaultSet)

file.print(i + " ");

file.print("}\nSet with simple numbers\n{ ");

for (int i : setSimple)

file.print(i + " ");

file.print("}\nSet with composite numbers\n{ ");

for (int i : setComposit)

file.print(i + " ");

file.println("}");

} catch (IOException e) {

isIncorrect = true;

System.err.println(ERRORS[Codes.IN\_OUT\_FILE\_EXCEPTION.ordinal()]);

}

} while (isIncorrect);

System.out.println("Writing is successfull");

}

static void outputInf(TreeSet<Integer> defaultSet, TreeSet<Integer> setSimple,

TreeSet<Integer> setComposit, Scanner input){

int choice = userChoice(input);

if (choice == 1) {

writeInConsole(defaultSet, setSimple, setComposit);

} else {

writeInFile(defaultSet, setSimple, setComposit, input);

}

}

public static void main(String[] args) {

Scanner input = new Scanner(System.in);

int[] borders;

TreeSet<Integer> defaultSet, setWithSimple, setWithComposit;

printInf();

borders = inputInf(input);

defaultSet = createSetWithBorders(borders);

setWithSimple = getSetOfSimple(defaultSet);

setWithComposit = getSetOfComposit(defaultSet, setWithSimple);

outputInf(defaultSet, setWithSimple, setWithComposit, input);

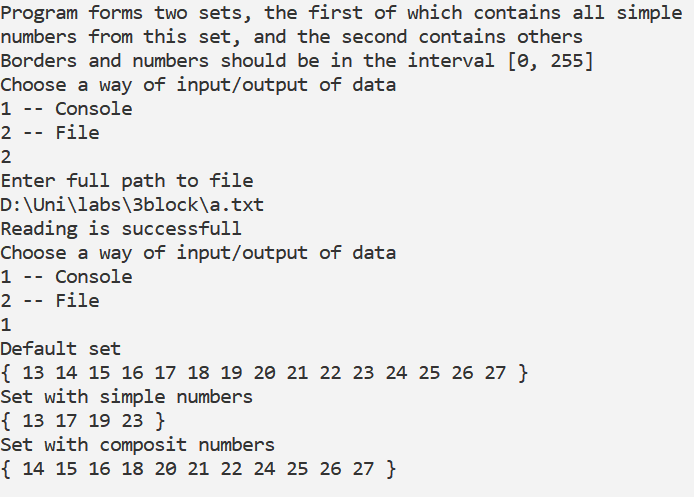
input.close();

}

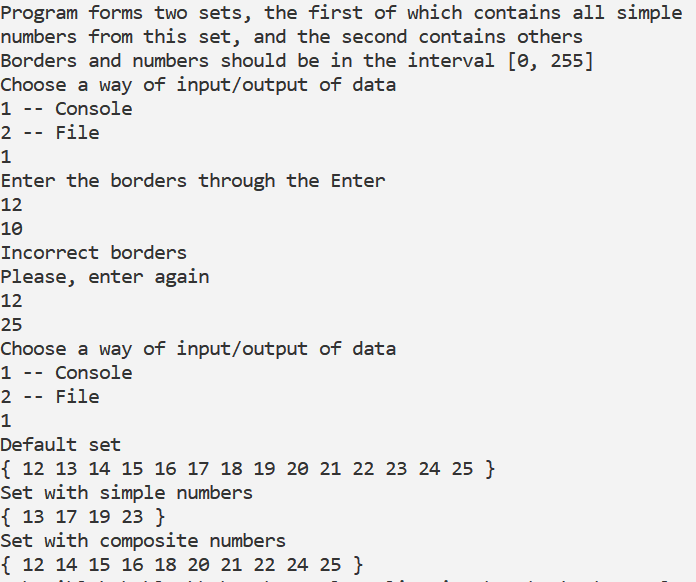
}

**Скриншоты:**

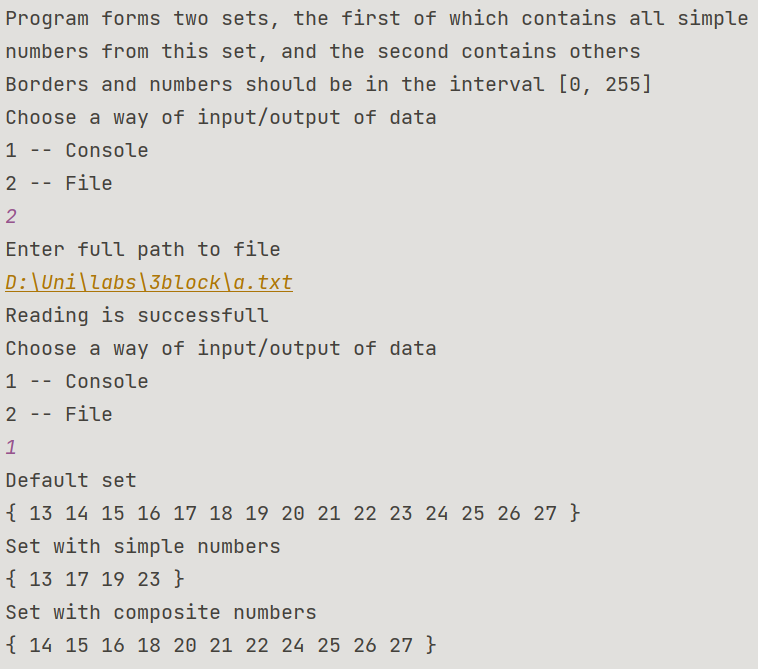
**Delphi:**

****

**C++:**

****

**Java:**

****

**Блок-схема:**





